# Logistic regression with Amazon food review dataset

In [1]:

```
import pandas as pd
import numpy as np
import pickle
#cleaned_data = pickle.load(open('cleaned.p','rb'))

cleaned_data = pickle.load(open('clen.p','rb')) ##
print(cleaned_data.shape)
print(type(cleaned_data))

cleaned_data=cleaned_data.iloc[:100000]
cleaned_data.shape
```

```
(250000, 11)
<class 'pandas.core.frame.DataFrame'>
```

Out[1]:

```
(100000, 11)
```

In [2]:

```
#Sort the data with accordance to time
cleaned_data.sort_values('Time',inplace=True)
cleaned_data.head(3)
```

Out[2]:

| | Id | ProductId | UserId | ProfileName | HelpfulnessNumerator | HelpfulnessDenominator | Score | |
|---|---|---|---|---|---|---|---|---|
| **138706** | 150524 | 0006641040 | ACITT7DI6IDDL | shari zychinski | 0 | 0 | positive | 93 |
| **138683** | 150501 | 0006641040 | AJ46FKXOVC7NR | Nicholas A Mesiano | 2 | 2 | positive | 94 |
| **417839** | 451856 | B00004CXX9 | AIUWLEQ1ADEG5 | Elizabeth Medina | 0 | 0 | positive | 94 |

In [3]:

```
# Find if the Y is imbalanced.
#positive_negative = cleaned_data['Score']
#print (positive_negative.value_counts())
#positive_negative.shape# --------------------found the Y is imbalanced

def partition(x):
    if x == 'negative':
        return '0'
    return '1'
actualScore = cleaned_data['Score']
positiveNegative = actualScore.map(partition)
cleaned_data['Score'] = positiveNegative
```

In [4]:

```
#positive_negative= cleaned_data['Score'].map(lambda x: 1 if int (x) is 'positive' else 0)
#print (positive_negative.value_counts())
positiveNegative = cleaned_data['Score']
print (positiveNegative.value_counts())
positiveNegative.shape

cleaned_data.head(5)
```

```
1    87730
0    12270
Name: Score, dtype: int64
```

Out[4]:

| | Id | ProductId | UserId | ProfileName | HelpfulnessNumerator | HelpfulnessDenominator | Score | |
|---|---|---|---|---|---|---|---|---|
| **138706** | 150524 | 0006641040 | ACITT7DI6IDDL | shari zychinski | 0 | 0 | 1 | 93 |
| **138683** | 150501 | 0006641040 | AJ46FKXOVC7NR | Nicholas A Mesiano | 2 | 2 | 1 | 94 |
| **417839** | 451856 | B00004CXX9 | AIUWLEQ1ADEG5 | Elizabeth Medina | 0 | 0 | 1 | 94 |
| **346055** | 374359 | B00004CI84 | A344SMIA5JECGM | Vincent P. Ross | 1 | 2 | 1 | 94 |
| **417838** | 451855 | B00004CXX9 | AJH6LUC1UT1ON | The Phantom of the Opera | 0 | 0 | 1 | 94 |

In [5]:

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer

X_tra, X_tes, y_train, y_test = train_test_split(cleaned_data['CleanedText'].values,positiveNegativ
e,test_size=0.3,shuffle=False)

#Implementing BAG of words
tf_idf_vect = TfidfVectorizer(ngram_range=(1,0),dtype=float)
X_tf = tf_idf_vect.fit_transform(X_tra)

# Standerdising the data
norm = StandardScaler(with_mean = False)
X_train = norm.fit_transform(X_tf)

# tfidf test
X_tfte = tf_idf_vect.transform(X_tes)

# Standerdising the data
X_test = norm.transform(X_tfte)
''' Use transform for follwing function (vectorizer and standerdisation) after main function
and not appligable for actual models'''
and not applicable for actual models
#uni_gram = CountVectorizer(ngram_range=(1,1))
#X_BOW = uni_gram.fit_transform(X_tra)
# Standerdising the data
#X_train2 = StandardScaler(with_mean = False).fit_transform(X_BOW)

#vectorizer for test data
#X_BOW1 = uni_gram.transform(X_tes)
# Standerdising the data
#X_test2 = StandardScaler(with_mean = False).fit_transform(X_BOW1)


from sklearn.model_selection import TimeSeriesSplit
tscv = TimeSeriesSplit(n_splits=10)
for train, cv in tscv.split(X_train):
    print(X_train[train].shape, X_train[cv].shape)
```

```
(6370, 41598) (6363, 41598)
(12733, 41598) (6363, 41598)
(19096, 41598) (6363, 41598)
(25459, 41598) (6363, 41598)
(31822, 41598) (6363, 41598)
(38185, 41598) (6363, 41598)
(44548, 41598) (6363, 41598)
(50911, 41598) (6363, 41598)
(57274, 41598) (6363, 41598)
(63637, 41598) (6363, 41598)
```

In [9]:

```
#GridSearch with default l2 norm
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import TimeSeriesSplit
from sklearn.metrics import make_scorer
from sklearn.metrics import f1_score

f1 = make_scorer(f1_score, pos_label='1')
tscv = TimeSeriesSplit(n_splits=3)
param_grid = {'C':[10**-4, 10**-2, 10**0, 10**2, 10**4]}
log = LogisticRegression(class_weight='balanced')
gsv = GridSearchCV(log,param_grid,cv=tscv,scoring=f1)
gsv.fit(X_train,y_train)

print("Best HyperParameter: ",gsv.best_params_)
print("Best f1: %.2f%%"%(gsv.best_score_*100))
```

```
Best HyperParameter:  {'C': 0.0001}
Best f1: 94.16%
```

In [10]:

```
print(gsv.best_estimator_)
```

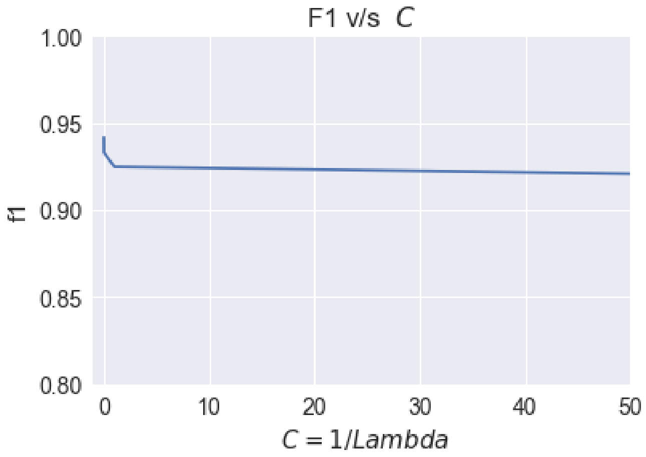
```
LogisticRegression(C=0.0001, class_weight='balanced', dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='ovr', n_jobs=1, penalty='l2', random_state=None,
          solver='liblinear', tol=0.0001, verbose=0, warm_start=False)
```

In [15]:

```
print(gsv.best_estimator_)
print("Best HyperParameter: ",gsv.best_params_)
print("Best f1: %.2f%%"%(gsv.best_score_*100))
import matplotlib.pyplot as plt
x=[]
y=[]
for a in gsv.grid_scores_:
    x.append(a[0]['C'])
    y.append(a[1])
plt.xlim(-1,50)
plt.ylim(0.8,1)
plt.xlabel(r"$\ C = 1/Lambda $",fontsize=15)
plt.ylabel("f1")
plt.title(r'F1 v/s $\ C$')
plt.plot(x,y)
plt.show()
```

```
LogisticRegression(C=0.0001, class_weight='balanced', dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='ovr', n_jobs=1, penalty='l2', random_state=None,
          solver='liblinear', tol=0.0001, verbose=0, warm_start=False)
Best HyperParameter:  {'C': 0.0001}
Best f1: 94.16%
```

```
C:\Users\admin\Anaconda3\lib\site-packages\sklearn\model_selection\_search.py:761:
DeprecationWarning: The grid_scores_ attribute was deprecated in version 0.18 in favor of the more
elaborate cv_results_ attribute. The grid_scores_ attribute will not be available from 0.20
  DeprecationWarning)
```



In [14]:

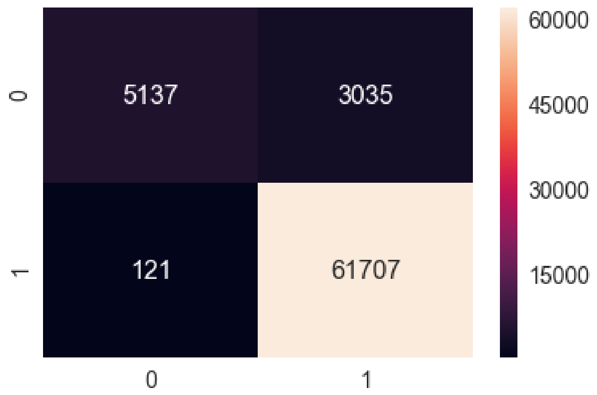
```
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.linear_model import LogisticRegression
import seaborn as sns



clf = LogisticRegression(C= 0.0001, penalty= 'l2')
clf.fit(X_train,y_train)
y_pred = clf.predict(X_train)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_train, y_pred)*100))
print("Non Zero weights:",np.count_nonzero(clf.coef_))
print("F1-Score on test set: %0.3f%%"%(f1_score(y_train, y_pred, pos_label='1')*100))
print (classification_report(y_train,y_pred))
result = confusion_matrix(y_train,y_pred)
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
print(result)
sns.set(font_scale=1.4)#for label size
sns.heatmap(result, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
Accuracy on test set: 95.491%
Non Zero weights: 41598
F1-Score on test set: 97.507%
             precision    recall  f1-score   support

          0       0.98      0.63      0.77      8172
          1       0.95      1.00      0.98     61828

avg / total       0.96      0.95      0.95     70000

Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]
```

```
[[ 5137  3035]
 [  121 61707]]
```

Out[14]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xc7619e8>
```



In [16]:

```
clf = LogisticRegression(C=1, penalty='l2')
model=clf.fit(X_train, y_train)
print("Non Zero weights:",np.count_nonzero(clf.coef_))
```

```
Non Zero weights: 41598
```

In [17]:

```
clf = LogisticRegression(C=1, penalty='l1')
model=clf.fit(X_train, y_train)
print("Non Zero weights:",np.count_nonzero(clf.coef_))
```

```
Non Zero weights: 15981
```

In [18]:

```
clf = LogisticRegression(C=100, penalty='l2')
model=clf.fit(X_train, y_train)
print("Non Zero weights:",np.count_nonzero(clf.coef_))
```

```
Non Zero weights: 41598
```

In [19]:

```
clf = LogisticRegression(C=100, penalty='l1')
model=clf.fit(X_train, y_train)
print("Non Zero weights:",np.count_nonzero(clf.coef_))
```

```
Non Zero weights: 18136
```

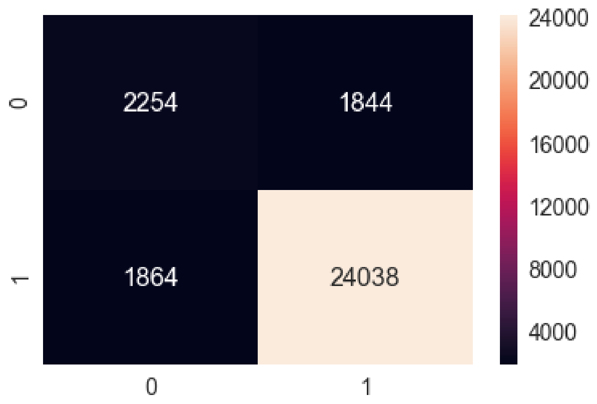
In [20]:

```
#test
clf = LogisticRegression(C=1, penalty='l1');
model=clf.fit(X_train, y_train);
pred = (model.predict(X_test))
```

In [21]:

```
print (classification_report(y_test,pred))
result = confusion_matrix(y_test,pred)
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
print(result)
sns.set(font_scale=1.4)#for label size
sns.set(font_scale=1.4)#for label size
sns.heatmap(result, annot=True,annot_kws={"size": 16}, fmt='g')

print("f1 on test set: %0.3f%%"%(f1_score(y_test, pred,pos_label='1')*100))
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, pred)*100))
```

```
             precision    recall  f1-score   support

          0       0.55      0.55      0.55      4098
          1       0.93      0.93      0.93     25902

avg / total       0.88      0.88      0.88     30000

Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]

[[ 2254  1844]
 [ 1864 24038]]
f1 on test set: 92.839%
Accuracy on test set: 87.640%
```



In [42]:

```
# Feature importance

all_feat = tf_idf_vect.get_feature_names()

feature_coefs = pd.DataFrame(
    data = list(zip(all_feat, model.coef_[0])),
    columns = ['-ve feature', 'coef'])

feature_imp = feature_coefs.sort_values(by='coef', ascending=True)
feature_imp.head(8)
```

Out[42]:

| | -ve feature | coef |
|---|---|---|
| **40796** | worst | -0.494986 |
| **10305** | disappoint | -0.460843 |
| **2196** | author | -0.416992 |
| **35729** | tast | -0.387144 |
| **2287** | aw | -0.364683 |
| **14110** | fragment | -0.353957 |
| **16501** | hard | -0.322843 |
| **34073** | stale | -0.317204 |

In [43]:

```
feature_coefs1 = pd.DataFrame(
    columns = ['+ve feature', 'coef'],
    data = list(zip(all_feat, model.coef_[0])))
```

```
feature_imp = feature_coefs1.sort_values(by='coef', ascending=False)
feature_imp.head(8)
```

Out[43]:

|  | +ve feature | coef |
|---|---|---|
| **15809** | great | 1.471587 |
| **3318** | best | 1.246960 |
| **21214** | love | 1.134374 |
| **41378** | yum | 0.993332 |
| **20488** | leg | 0.947100 |
| **16758** | heartili | 0.886589 |
| **41391** | yummi | 0.837720 |
| **9628** | delici | 0.832593 |

In [178]:

```
# Perturbation tech
from scipy.sparse import find

clf = LogisticRegression(C=0.0001, penalty='l2')
clf.fit(X_train, y_train)
weights_before = clf.coef_

epsilon = np.random.normal(scale=0.1)
print ('noise adding to weight vector is = {0}'.format(epsilon))
X_train1 = X_train
a,b = np.nonzero(X_train1)
X_train1[a,b] = X_train[a,b]+epsilon
clf.fit(X_train1,y_train)
weights_after = clf.coef_
print ('weights_after adding noise = {0}'.format(weights_after))
```

```
noise adding to weight vector is = -0.05678220906943951
weights_after adding noise = [[ 1.40735563e-03  1.17262635e-03  1.45049363e-03 ... -5.78981902e-03
   2.22507124e-04  5.26017521e-05]]
```

In [179]:

```
weights_diff = abs(weights_before - weights_after)
per_weight_diff1=[]
per_weight_diff1 = weights_diff-0.3*abs (weights_before)
(per_weight_diff1[per_weight_diff1>0].size)/weights_before.size*100
```

Out[179]:

```
0.2497055359246172
```

In [181]:

```
import numpy
weights_diff.shape
value1 = numpy.where( weights_diff > 0.3 )# numpy for above 30%
print(value1)
weights_diff[ numpy.where( weights_diff > 0.01 ) ] # find the exact value above 0.01
```

```
(array([], dtype=int64), array([], dtype=int64))
```

Out[181]:

```
array([], dtype=float64)
```

# Random search model

In [22]:

```
# RandomsearchCV with default l2 norm
#from sklearn.model_selection import TimeSeriesSplit
#print(X_train[train].shape, X_train[cv].shape)
from sklearn.model_selection import TimeSeriesSplit
from sklearn.model_selection import RandomizedSearchCV
from sklearn.linear_model import LogisticRegression


f1 = make_scorer(f1_score, pos_label='1')
tscv = TimeSeriesSplit(n_splits=3)
#clf = LogisticRegression(class_weight='balanced',penalty='l2)
#params we need to try on classifier
param_grid = {'C':[10**-4, 10**-2, 10**0, 10**2, 10**4]}

rsv = RandomizedSearchCV(LogisticRegression(class_weight='balanced',penalty='l2'),
param_grid,n_iter=5,cv=tscv,scoring=f1)
rsv.fit(X_train,y_train)

#savetofile(gsv,"Log Reg/gsv_uni")
print("Best HyperParameter: ",rsv.best_params_)
print("Best F1 score: %.2f%%"%(rsv.best_score_*100))
```

```
Best HyperParameter:  {'C': 0.0001}
Best F1 score: 94.16%
```

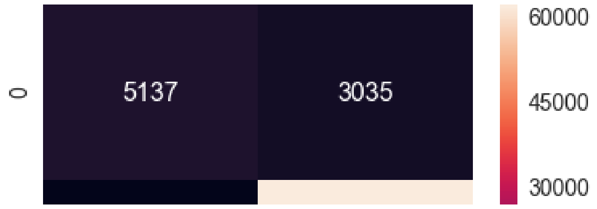
In [23]:

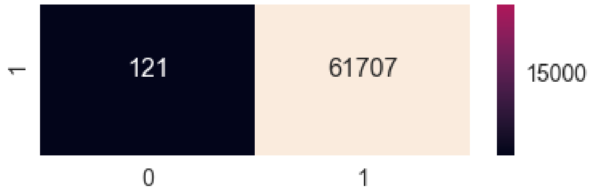
```
clf = LogisticRegression(C= 0.0001, penalty= 'l2')
clf.fit(X_train,y_train)
y_pred = clf.predict(X_train)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_train, y_pred)*100))
print("Non Zero weights:",np.count_nonzero(clf.coef_))
print("F1-Score on test set: %0.3f%%"%(f1_score(y_train, y_pred, pos_label='1')*100))
print (classification_report(y_train,y_pred))
result = confusion_matrix(y_train,y_pred)
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
print(result)
sns.set(font_scale=1.4)#for label size
sns.heatmap(result, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
Accuracy on test set: 95.491%
Non Zero weights: 41598
F1-Score on test set: 97.507%
             precision    recall  f1-score   support

          0       0.98      0.63      0.77      8172
          1       0.95      1.00      0.98     61828

avg / total       0.96      0.95      0.95     70000

Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]

[[ 5137  3035]
 [  121 61707]]
```

Out[23]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xd8eaac8>
```





In [24]:

```
clf = LogisticRegression(C=1, penalty='l2')
model=clf.fit(X_train, y_train)
print("Non Zero weights:",np.count_nonzero(clf.coef_))
```

```
Non Zero weights: 41598
```

In [25]:

```
clf = LogisticRegression(C=1, penalty='l1')
model=clf.fit(X_train, y_train)
print("Non Zero weights:",np.count_nonzero(clf.coef_))
```

```
Non Zero weights: 15942
```

In [26]:

```
clf = LogisticRegression(C=100, penalty='l2')
model=clf.fit(X_train, y_train)
print("Non Zero weights:",np.count_nonzero(clf.coef_))
```

```
Non Zero weights: 41598
```

In [27]:

```
clf = LogisticRegression(C=100, penalty='l1')
model=clf.fit(X_train, y_train)
print("Non Zero weights:",np.count_nonzero(clf.coef_))
```

```
Non Zero weights: 18125
```

In [28]:

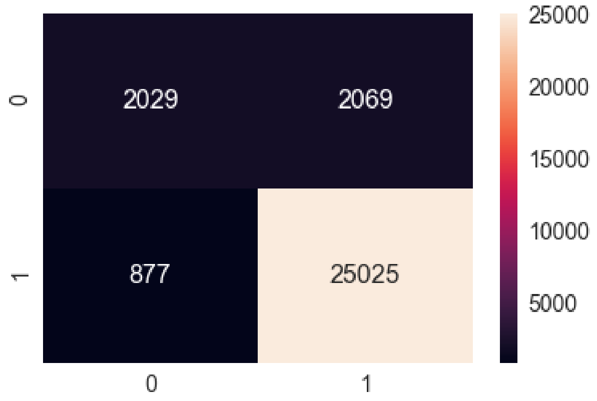
```
clf = LogisticRegression(C=0.001, penalty='l2');
model=clf.fit(X_train, y_train);
pred = (model.predict(X_test))
```

In [29]:

```
print (classification_report(y_test,pred))
result = confusion_matrix(y_test,pred)
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
print(result)
sns.set(font_scale=1.4)#for label size
sns.heatmap(result, annot=True,annot_kws={"size": 16}, fmt='g')

print("f1 on test set: %0.3f%%"%(f1_score(y_test, pred, pos_label='1')*100))
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, pred)*100))
```

```
             precision    recall  f1-score   support

          0       0.70      0.50      0.58      4098
          1       0.92      0.97      0.94     25902

avg / total       0.89      0.90      0.89     30000

Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]
```

```
[[ 2029  2069]
 [  877 25025]]
f1 on test set: 94.441%
Accuracy on test set: 90.180%
```



| LR with Grid search CV | LR with Random search CV |
|---|---|
| 1. Hyper parameter is 0.001 | Hyper parameter is 0.001 |
| 2. F1 score of Train data = 97.5% | F1 score of Train data = 97.5% |
| 3. F1 score of Test data | F1 score of Test data |
| 92.85% | 94.4% |

# BOW Logistic regression

## Grid search

In [30]:

```
#count_vect = CountVectorizer(1,1)
#count_vect.fit_transform(final['CleanedText'].values)

#Implementing BAG of words
bi_gram = CountVectorizer(1,0)
X_BOW = bi_gram.fit_transform(X_tra)
# Standerdising the data
norm = StandardScaler(with_mean = False)
X_train = norm.fit_transform(X_BOW)
#X_train = preprocessing.normalize(X_BOW)

#vectorizer for test data
X_BOW1 = bi_gram.transform(X_tes)
# Standerdising the data
X_test = norm.transform(X_BOW1)

from sklearn.model_selection import TimeSeriesSplit

for train, cv in tscv.split(X_train):
    print(X_train[train].shape, X_train[cv].shape)
```

```
C:\Users\admin\Anaconda3\lib\site-packages\sklearn\utils\validation.py:475: DataConversionWarning:
Data with input dtype int64 was converted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
C:\Users\admin\Anaconda3\lib\site-packages\sklearn\utils\validation.py:475: DataConversionWarning:
Data with input dtype int64 was converted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
```

```
(17500, 41598) (17500, 41598)
(35000, 41598) (17500, 41598)
(52500, 41598) (17500, 41598)
```

```
C:\Users\admin\Anaconda3\lib\site-packages\sklearn\utils\validation.py:475: DataConversionWarning:
Data with input dtype int64 was converted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
```

In [31]:

```
f1 = make_scorer(f1_score, pos_label='1')
tscv = TimeSeriesSplit(n_splits=3)
param_grid = {'C':[10**-4, 10**-2, 10**0, 10**2, 10**4]}
log = LogisticRegression(class_weight='balanced')
gsv = GridSearchCV(log,param_grid,cv=tscv,scoring=f1)
gsv.fit(X_train,y_train)

print("Best HyperParameter: ",gsv.best_params_)
print("Best f1: %.2f%%"%(gsv.best_score_*100))
```

```
Best HyperParameter:  {'C': 0.0001}
Best f1: 94.61%
```

In [32]:

```
print(gsv.best_estimator_)
```

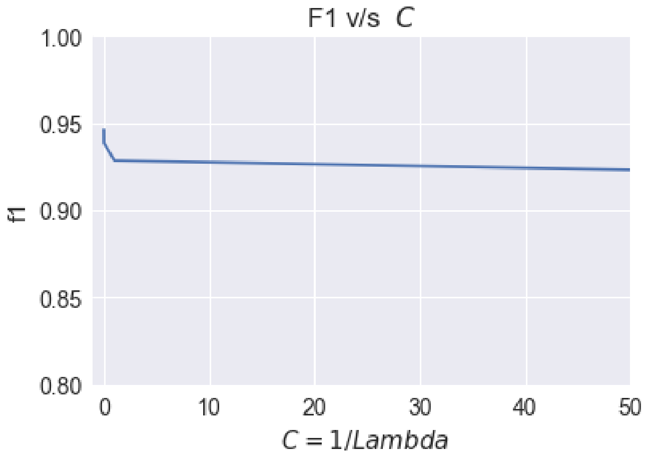
```
LogisticRegression(C=0.0001, class_weight='balanced', dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='ovr', n_jobs=1, penalty='l2', random_state=None,
          solver='liblinear', tol=0.0001, verbose=0, warm_start=False)
```

In [33]:

```
print("Best HyperParameter: ",gsv.best_params_)
print("Best f1: %.2f%%"%(gsv.best_score_*100))
import matplotlib.pyplot as plt
x=[]
y=[]
for a in gsv.grid_scores_:
    x.append(a[0]['C'])
    y.append(a[1])
plt.xlim(-1,50)
plt.ylim(0.8,1)
plt.xlabel(r"$\ C = 1/Lambda $",fontsize=15)
plt.ylabel("f1")
plt.title(r'F1 v/s $\ C$')
plt.plot(x,y)
plt.show()
```

```
Best HyperParameter:  {'C': 0.0001}
Best f1: 94.61%
```

```
C:\Users\admin\Anaconda3\lib\site-packages\sklearn\model_selection\_search.py:761:
DeprecationWarning: The grid_scores_ attribute was deprecated in version 0.18 in favor of the more
elaborate cv_results_ attribute. The grid_scores_ attribute will not be available from 0.20
  DeprecationWarning)
```



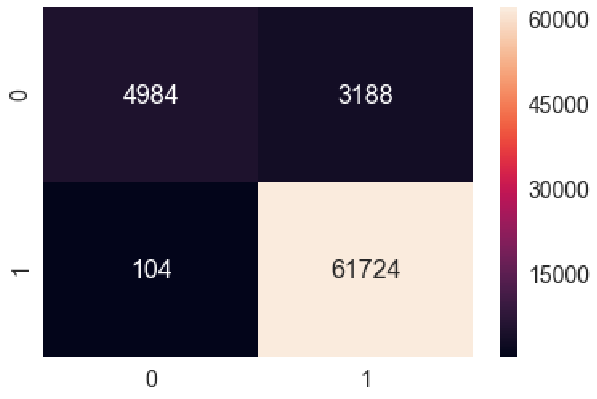
In [34]:

```
clf = LogisticRegression(C= 0.0001, penalty= 'l2')
clf.fit(X_train,y_train)
y_pred = clf.predict(X_train)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_train, y_pred)*100))
print("Non Zero weights:",np.count_nonzero(clf.coef_))
print("F1-Score on test set: %0.3f%%"%(f1_score(y_train, y_pred, pos_label='1')*100))
print (classification_report(y_train,y_pred))
result = confusion_matrix(y_train,y_pred)
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
print(result)
sns.set(font_scale=1.4)#for label size
sns.heatmap(result, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
Accuracy on test set: 95.297%
Non Zero weights: 41598
F1-Score on test set: 97.403%
             precision    recall  f1-score   support

          0       0.98      0.61      0.75      8172
          1       0.95      1.00      0.97     61828

avg / total       0.95      0.95      0.95     70000

Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]

[[ 4984  3188]
 [  104 61724]]
```

Out[34]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xdc78e10>
```



In [38]:

```
clf = LogisticRegression(C=1, penalty='l2')
model=clf.fit(X_train, y_train)
print("Non Zero weights:",np.count_nonzero(clf.coef_))
```

```
Non Zero weights: 41598
```

In [39]:

```
clf = LogisticRegression(C=1, penalty='l1')
model=clf.fit(X_train, y_train)
print("Non Zero weights:",np.count_nonzero(clf.coef_))
```

```
Non Zero weights: 13909
```

In [40]:

```
clf = LogisticRegression(C=100, penalty='l2')
model=clf.fit(X_train, y_train)
print("Non Zero weights:",np.count_nonzero(clf.coef_))
```

```
Non Zero weights: 41598
Non Zero weights: 41598
```

In [21]:

```
clf = LogisticRegression(C=100, penalty='l1')
model=clf.fit(X_train, y_train)
print("Non Zero weights:",np.count_nonzero(clf.coef_))
```

```
Non Zero weights: 18201
```

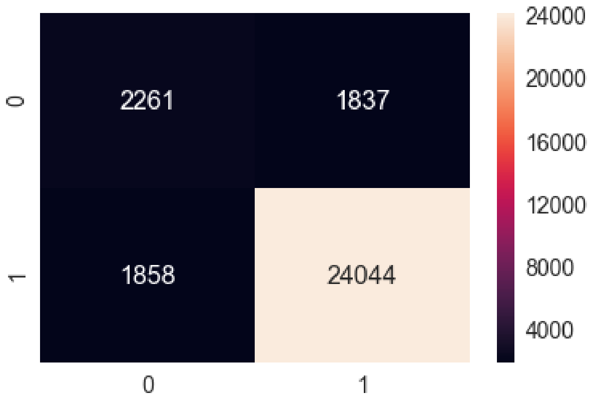
In [22]:

```
#test
clf = LogisticRegression(C=1, penalty='l1');
model=clf.fit(X_train, y_train);
pred = (model.predict(X_test))
```

In [23]:

```
print (classification_report(y_test,pred))
result = confusion_matrix(y_test,pred)
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
print(result)
sns.set(font_scale=1.4)#for label size
sns.heatmap(result, annot=True,annot_kws={"size": 16}, fmt='g')

print("f1 on test set: %0.3f%%"%(f1_score(y_test, pred, pos_label='1')*100))
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, pred)*100))
```

```
             precision    recall  f1-score   support

          0       0.55      0.55      0.55      4098
          1       0.93      0.93      0.93     25902

avg / total       0.88      0.88      0.88     30000

Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]

[[ 2261  1837]
 [ 1858 24044]]
f1 on test set: 92.864%
Accuracy on test set: 87.683%
```



# Random search model

In [12]:

```
from sklearn.metrics import make_scorer
from sklearn.metrics import f1_score
from sklearn.model_selection import RandomizedSearchCV
from sklearn.linear_model import LogisticRegression

f1 = make_scorer(f1_score, pos_label='1')
```

```
f1 = make_scorer(f1_score, pos_label='1')
tscv = TimeSeriesSplit(n_splits=3)
#clf = LogisticRegression(class_weight='balanced',penalty='l2)
#params we need to try on classifier
param_grid = {'C':[10**-4, 10**-2, 10**0, 10**2, 10**4]}

rsv = RandomizedSearchCV(LogisticRegression(class_weight='balanced',penalty='l2'),
param_grid,n_iter=5,cv=tscv,scoring=f1)
rsv.fit(X_train,y_train)

#savetofile(gsv,"Log Reg/gsv_uni")
print("Best HyperParameter: ",rsv.best_params_)
print("Best F1 score: %.2f%%"%(rsv.best_score_*100))
```

```
Best HyperParameter:  {'C': 0.0001}
Best F1 score: 94.16%
```

In [14]:

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.linear_model import LogisticRegression
import seaborn as sns

clf = LogisticRegression(C= 0.0001, penalty= 'l2')
clf.fit(X_train,y_train)
y_pred = clf.predict(X_train)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_train, y_pred)*100))
print("Non Zero weights:",np.count_nonzero(clf.coef_))
print("F1-Score on test set: %0.3f%%"%(f1_score(y_train, y_pred, pos_label='1')*100))
print (classification_report(y_train,y_pred))
result = confusion_matrix(y_train,y_pred)
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
print(result)
sns.set(font_scale=1.4)#for label size
sns.heatmap(result, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
Accuracy on test set: 95.491%
Non Zero weights: 41598
F1-Score on test set: 97.507%
             precision    recall  f1-score   support

          0       0.98      0.63      0.77      8172
          1       0.95      1.00      0.98     61828

avg / total       0.96      0.95      0.95     70000

Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]

[[ 5137  3035]
 [  121 61707]]
```

Out[14]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xc8b03c8>
```

In [15]:

```
clf = LogisticRegression(C=1, penalty='l2')
model=clf.fit(X_train, y_train)
print("Non Zero weights:",np.count_nonzero(clf.coef_))
```

```
Non Zero weights: 41598
```

In [16]:

```
clf = LogisticRegression(C=100, penalty='l1')
```

```
model=clf.fit(X_train, y_train)
print("Non Zero weights:",np.count_nonzero(clf.coef_))
```

```
Non Zero weights: 18155
```

In [17]:

```
clf = LogisticRegression(C=100, penalty='l2')
model=clf.fit(X_train, y_train)
print("Non Zero weights:",np.count_nonzero(clf.coef_))
```

```
Non Zero weights: 41598
```

In [18]:

```
clf = LogisticRegression(C=1, penalty='l2')
model=clf.fit(X_train, y_train)
print("Non Zero weights:",np.count_nonzero(clf.coef_))
```

```
Non Zero weights: 41598
```

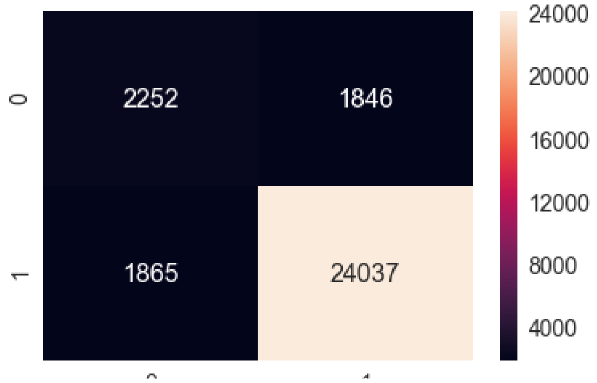
In [19]:

```
#test
clf = LogisticRegression(C=1, penalty='l1');
model=clf.fit(X_train, y_train);
pred = (model.predict(X_test))
```

In [20]:

```
print (classification_report(y_test,pred))
result = confusion_matrix(y_test,pred)
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
print(result)
sns.set(font_scale=1.4)#for label size
sns.heatmap(result, annot=True,annot_kws={"size": 16}, fmt='g')

print("f1 on test set: %0.3f%%"%(f1_score(y_test, pred, pos_label='1')*100))
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, pred)*100))
```

```
             precision    recall  f1-score   support

          0       0.55      0.55      0.55      4098
          1       0.93      0.93      0.93     25902

avg / total       0.88      0.88      0.88     30000

Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]

[[ 2252  1846]
 [ 1865 24037]]
f1 on test set: 92.834%
Accuracy on test set: 87.630%
```



0 1

| **LR with Grid search CV** | **LR with Random search CV** |
|---|---|
| 1. Hyper parameter is 0.001 | Hyper parameter is 0.001 |
| 2. F1 score of Train data = 97.5% | F1 score of Train data = 97.5% |
| 3. F1 score of Test data | F1 score of Test data |
| 92.85% | 92.83% |

# Logistic Regression with average word2vec

In [74]:

```
import pandas as pd
import numpy as np
import pickle


cleaned_data = pickle.load(open('drive/My Drive/Colab Notebooks/new.p','rb')) ##
print(cleaned_data.shape)
print(type(cleaned_data))

cleaned_data=cleaned_data.iloc[:100000]
cleaned_data.shape
```

```
(250000, 11)
<class 'pandas.core.frame.DataFrame'>
```

Out[74]:

```
(100000, 11)
```

In [75]:

```
#Sort the data with accordance to time
cleaned_data.sort_values('Time',inplace=True)
cleaned_data.head(3)
```

Out[75]:

| | Id | ProductId | UserId | ProfileName | HelpfulnessNumerator | HelpfulnessDenominator | Score | Time | S |
|---|---|---|---|---|---|---|---|---|---|
| **138706** | 150524 | 0006641040 | ACITT7DI6IDDL | shari zychinski | 0 | 0 | positive | 939340800 | edı |
| **138683** | 150501 | 0006641040 | AJ46FKXOVC7NR | Nicholas A Mesiano | 2 | 2 | positive | 940809600 | Th<br>grea<br>sp |
| **417839** | 451856 | B00004CXX9 | AIUWLEQ1ADEG5 | Elizabeth Medina | 0 | 0 | positive | 944092800 | Ente |

In [76]:

```
!pip install gensim
```

```
Requirement already satisfied: gensim in /usr/local/lib/python3.6/dist-packages (3.6.0)
Requirement already satisfied: scipy>=0.18.1 in /usr/local/lib/python3.6/dist-packages (from
gensim) (0.19.1)
Requirement already satisfied: six>=1.5.0 in /usr/local/lib/python3.6/dist-packages (from gensim)
(1.11.0)
Requirement already satisfied: numpy>=1.11.3 in /usr/local/lib/python3.6/dist-packages (from
gensim) (1.14.6)
Requirement already satisfied: smart-open>=1.2.1 in /usr/local/lib/python3.6/dist-packages (from
gensim) (1.7.1)
Requirement already satisfied: boto3 in /usr/local/lib/python3.6/dist-packages (from smart-
open>=1.2.1->gensim) (1.9.23)
Requirement already satisfied: requests in /usr/local/lib/python3.6/dist-packages (from smart-
open>=1.2.1->gensim) (2.18.4)
Requirement already satisfied: boto>=2.32 in /usr/local/lib/python3.6/dist-packages (from smart-
open>=1.2.1->gensim) (2.49.0)
```

```
Requirement already satisfied: bz2file in /usr/local/lib/python3.6/dist-packages (from smart-
open>=1.2.1->gensim) (0.98)
Requirement already satisfied: botocore<1.13.0,>=1.12.23 in /usr/local/lib/python3.6/dist-packages
(from boto3->smart-open>=1.2.1->gensim) (1.12.23)
Requirement already satisfied: jmespath<1.0.0,>=0.7.1 in /usr/local/lib/python3.6/dist-packages
(from boto3->smart-open>=1.2.1->gensim) (0.9.3)
Requirement already satisfied: s3transfer<0.2.0,>=0.1.10 in /usr/local/lib/python3.6/dist-packages
(from boto3->smart-open>=1.2.1->gensim) (0.1.13)
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in /usr/local/lib/python3.6/dist-packages
(from requests->smart-open>=1.2.1->gensim) (3.0.4)
Requirement already satisfied: urllib3<1.23,>=1.21.1 in /usr/local/lib/python3.6/dist-packages
(from requests->smart-open>=1.2.1->gensim) (1.22)
Requirement already satisfied: idna<2.7,>=2.5 in /usr/local/lib/python3.6/dist-packages (from
requests->smart-open>=1.2.1->gensim) (2.6)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.6/dist-packages (from
requests->smart-open>=1.2.1->gensim) (2018.8.24)
Requirement already satisfied: python-dateutil<3.0.0,>=2.1; python_version >= "2.7" in
/usr/local/lib/python3.6/dist-packages (from botocore<1.13.0,>=1.12.23->boto3->smart-open>=1.2.1->
gensim) (2.5.3)
Requirement already satisfied: docutils>=0.10 in /usr/local/lib/python3.6/dist-packages (from
botocore<1.13.0,>=1.12.23->boto3->smart-open>=1.2.1->gensim) (0.14)
```

In [0]:

```
from sklearn.model_selection import train_test_split
X_tra, X_tes, y_train, y_test = train_test_split(cleaned_data['CleanedText'],cleaned_data['Score'].
values,test_size=0.3,shuffle=False)
```

In [0]:

```
sent_of_train=[]
for sent in X_tra:
    sent_of_train.append(sent.split())

#sent_of_train

sent_of_test=[]
for sent in X_tes:
    sent_of_test.append(sent.split())
```

In [80]:

```

from gensim.models import Word2Vec
w2v_model=Word2Vec(sent_of_train,min_count=3,size=50, workers=4)# words which occurs 3 times; 50
dimensions

w2v_words = list(w2v_model.wv.vocab)
print("number of words that occured minimum 3 times ",len(w2v_words))


from gensim.models import Word2Vec
w2v_model=Word2Vec(sent_of_test,min_count=3,size=50, workers=4)# words which occurs 3 times; 50
dimensions

w2v_words = list(w2v_model.wv.vocab)
print("number of words that occured minimum 3 times ",len(w2v_words))
```

```
number of words that occured minimum 3 times  14278
number of words that occured minimum 3 times  9641
```

In [81]:

```
# compute average word2vec for each review for X_train .
train_vectors = []
for sent in sent_of_train:
    sent_vec = np.zeros(50)
    cnt_words =0;
    for word in sent:
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    train_vectors.append(sent_vec)


# compute average word2vec for each review for X_test .
test_vectors = []
for sent in sent_of_test:
    sent_vec = np.zeros(50)
    cnt_words =0;
    for word in sent:
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    test_vectors.append(sent_vec)


X_train =np.asarray(train_vectors)
X_test =np.asarray(test_vectors)

X_train.shape,X_test.shape
```

Out[81]:

```
((70000, 50), (30000, 50))
```

In [82]:

```
# Data-preprocessing: Standardizing the data

sc = StandardScaler(with_mean = False)
X_train1 = sc.fit_transform(X_train)
X_test1 = sc.transform(X_test)


tscv = TimeSeriesSplit(n_splits=4)
for train, cv in tscv.split(X_train1):
    print(X_train1[train].shape, X_train1[cv].shape)
```

```
(14000, 50) (14000, 50)
(28000, 50) (14000, 50)
(42000, 50) (14000, 50)
(56000, 50) (14000, 50)
```

In [107]:

```
#GridSearch with default l2 norm
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import TimeSeriesSplit
from sklearn.metrics import make_scorer
from sklearn.metrics import f1_score

f1 = make_scorer(f1_score, pos_label='positive')
tscv = TimeSeriesSplit(n_splits=3)
param_grid = {'C':[10**-4, 10**-2, 10**0, 10**2, 10**4]}
log = LogisticRegression(class_weight='balanced')
gsv = GridSearchCV(log,param_grid,cv=tscv,scoring=f1)
gsv.fit(X_train1,y_train)

print("Best HyperParameter: ",gsv.best_params_)
print("Best f1: %.2f%%"%(gsv.best_score_*100))
```

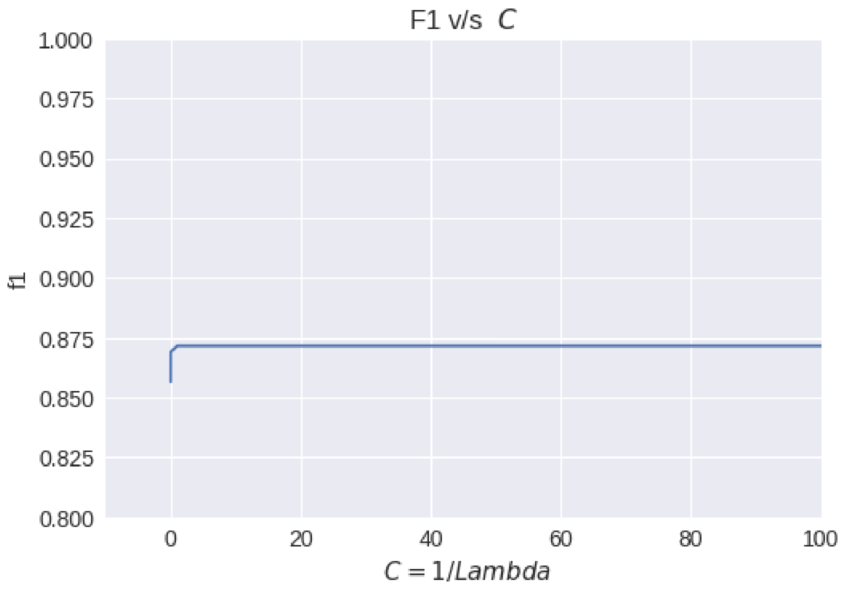
```
Best HyperParameter:  {'C': 1}
Best f1: 87.16%
```

In [110]:

```
print(gsv.best_estimator_)
print("Best HyperParameter: ",gsv.best_params_)
print("Best f1: %.2f%%"%(gsv.best_score_*100))
import matplotlib.pyplot as plt
x=[]
y=[]
for a in gsv.grid_scores_:
    x.append(a[0]['C'])
    y.append(a[1])
plt.xlim(-10,100)
plt.ylim(0.8,1)
plt.xlabel(r"$\ C = 1/Lambda $",fontsize=15)
plt.ylabel("f1")
plt.title(r'F1 v/s $\ C$')
plt.plot(x,y)
plt.show()
```

```
LogisticRegression(C=1, class_weight='balanced', dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='ovr', n_jobs=1, penalty='l2', random_state=None,
          solver='liblinear', tol=0.0001, verbose=0, warm_start=False)
Best HyperParameter:  {'C': 1}
Best f1: 87.16%
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_search.py:762: DeprecationWarning:
The grid_scores_ attribute was deprecated in version 0.18 in favor of the more elaborate
cv_results_ attribute. The grid_scores_ attribute will not be available from 0.20
  DeprecationWarning)
```

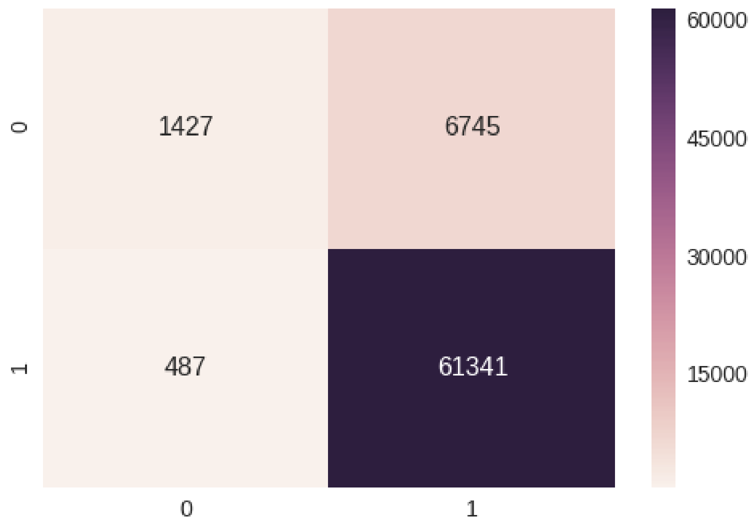


In [111]:

```
log = LogisticRegression(class_weight='balanced',C=1000)
log.fit(X_train1,y_train)

pred = (model.predict(X_train1))
print (classification_report(y_train,pred))
result = confusion_matrix(y_train,pred)
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
print(result)
sns.set(font_scale=1.4)#for label size
sns.heatmap(result, annot=True,annot_kws={"size": 16}, fmt='g')

print("f1 on test set: %0.3f%%"%(f1_score(y_train, pred,pos_label='positive')*100))
print("Non Zero weights:",np.count_nonzero(clf.coef_))
```

```
             precision    recall  f1-score   support

   negative       0.75      0.17      0.28      8172
   positive       0.90      0.99      0.94     61828

avg / total       0.88      0.90      0.87     70000
```

```
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]

[[ 1427  6745]
 [  487 61341]]
f1 on test set: 94.433%
Non Zero weights: 50
```



In [112]:

```
clf = LogisticRegression(C=100, penalty='l2')
model=clf.fit(X_train, y_train)
print("Non Zero weights:",np.count_nonzero(clf.coef_))
```

```
Non Zero weights: 50
```

In [113]:

```
clf = LogisticRegression(C=100, penalty='l1')
model=clf.fit(X_train, y_train)
print("Non Zero weights:",np.count_nonzero(clf.coef_))
```

```
Non Zero weights: 50
```

In [114]:

```
clf = LogisticRegression(C=0.01, penalty='l2')
model=clf.fit(X_train, y_train)
print("Non Zero weights:",np.count_nonzero(clf.coef_))
```

```
Non Zero weights: 50
```

In [115]:

```
clf = LogisticRegression(C=0.01, penalty='l1')
model=clf.fit(X_train, y_train)
print("Non Zero weights:",np.count_nonzero(clf.coef_))
```

```
Non Zero weights: 29
```

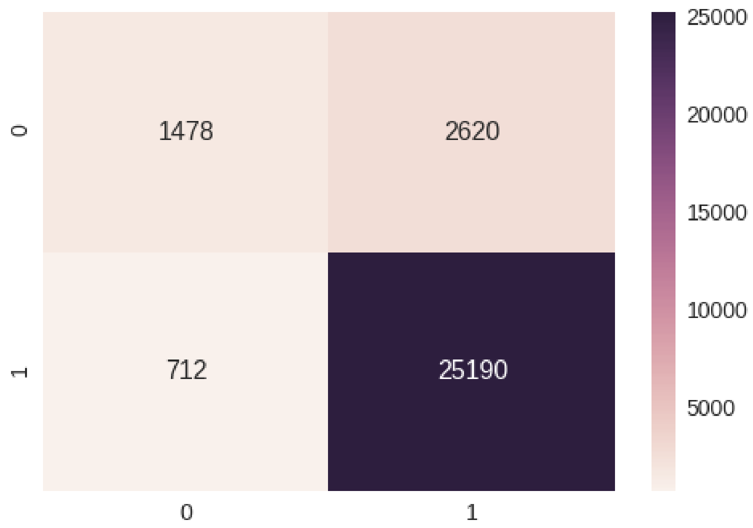
In [0]:

```

```

In [116]:

```
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
import seaborn as sns

pred = (model.predict(X_test1))
print (classification_report(y_test,pred))
result = confusion_matrix(y_test,pred)
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
print(result)
sns.set(font_scale=1.4)#for label size
sns.heatmap(result, annot=True,annot_kws={"size": 16}, fmt='g')

print("f1 on test set: %0.3f%%"%(f1_score(y_test, pred,pos_label='positive')*100))
```

```
             precision    recall  f1-score   support

   negative       0.67      0.36      0.47      4098
   positive       0.91      0.97      0.94     25902

avg / total       0.87      0.89      0.87     30000

Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]

[[ 1478  2620]
 [  712 25190]]
f1 on test set: 93.797%
```



## Random search

In [117]:

```
from sklearn.model_selection import RandomizedSearchCV

f1 = make_scorer(f1_score, pos_label='positive')
tscv = TimeSeriesSplit(n_splits=3)
#clf = LogisticRegression(class_weight='balanced',penalty='l2)
#params we need to try on classifier
param_grid = {'C':[10**-4, 10**-2, 10**0, 10**2, 10**4]}

rsv = RandomizedSearchCV(LogisticRegression(class_weight='balanced',penalty='l2'),
param_grid,n_iter=5,cv=tscv,scoring=f1)
rsv.fit(X_train1,y_train)

#savetofile(gsv,"Log Reg/gsv_uni")
print("Best HyperParameter: ",rsv.best_params_)
print("Best F1 score: %.2f%%"%(rsv.best_score_*100))
```

```
Best HyperParameter:  {'C': 1}
Best F1 score: 87.16%
```

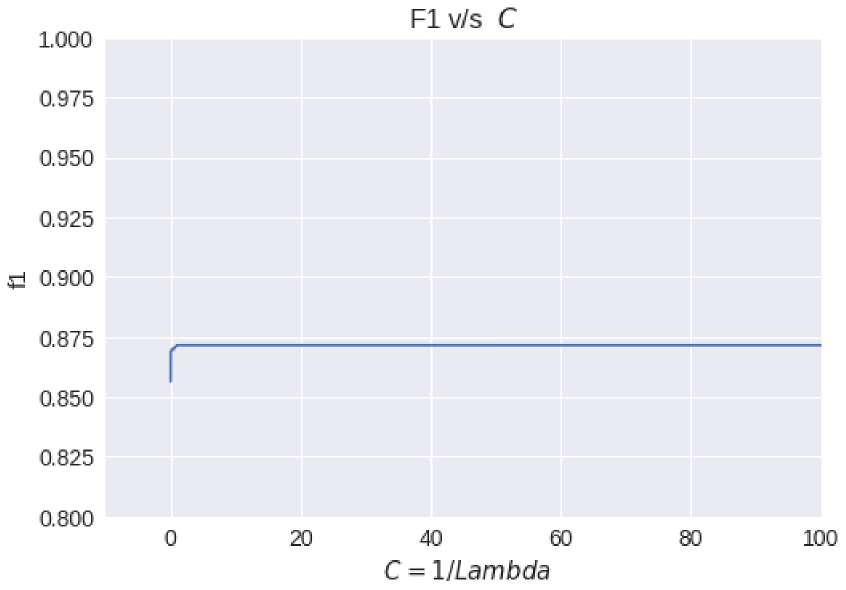
In [118]:

In [118]:

```
print(rsv.best_estimator_)
print("Best HyperParameter: ",rsv.best_params_)
print("Best f1: %.2f%%"%(rsv.best_score_*100))
import matplotlib.pyplot as plt
x=[]
y=[]
for a in rsv.grid_scores_:
    x.append(a[0]['C'])
    y.append(a[1])
plt.xlim(-10,100)
plt.ylim(0.8,1)
plt.xlabel(r"$\ C = 1/Lambda $",fontsize=15)
plt.ylabel("f1")
plt.title(r'F1 v/s $\ C$')
plt.plot(x,y)
plt.show()
```

```
LogisticRegression(C=1, class_weight='balanced', dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='ovr', n_jobs=1, penalty='l2', random_state=None,
          solver='liblinear', tol=0.0001, verbose=0, warm_start=False)
Best HyperParameter:  {'C': 1}
Best f1: 87.16%
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_search.py:762: DeprecationWarning:
The grid_scores_ attribute was deprecated in version 0.18 in favor of the more elaborate
cv_results_ attribute. The grid_scores_ attribute will not be available from 0.20
  DeprecationWarning)
```



In [123]:

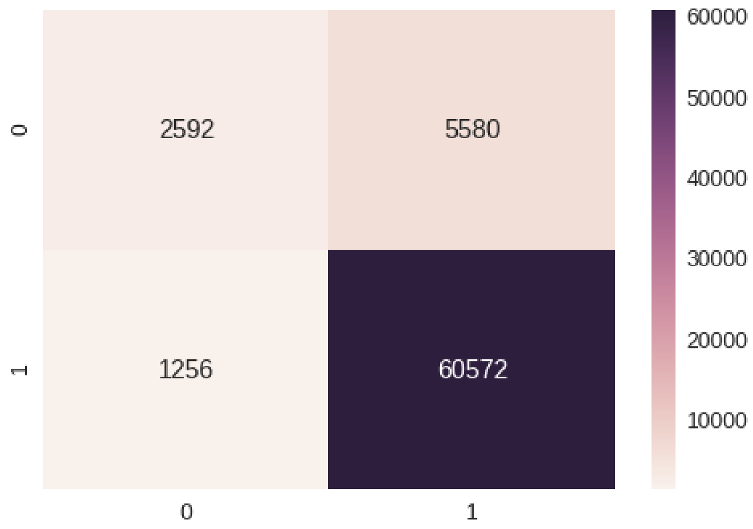
```
clf = LogisticRegression(C= 1, penalty= 'l2')
clf.fit(X_train1,y_train)
y_pred = clf.predict(X_train1)

print("Non Zero weights:",np.count_nonzero(clf.coef_))
print("F1-Score on test set: %0.3f%%"%(f1_score(y_train, y_pred, pos_label='positive')*100))
print (classification_report(y_train,y_pred))
result = confusion_matrix(y_train,y_pred)
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
print(result)
sns.set(font_scale=1.4)#for label size
sns.heatmap(result, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
Non Zero weights: 50
F1-Score on test set: 94.659%
             precision    recall  f1-score   support

   negative       0.67      0.32      0.43      8172
   positive       0.92      0.98      0.95     61828
```

```
avg / total       0.89      0.90      0.89     70000

Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]

[[ 2592  5580]
 [ 1256 60572]]
```

Out[123]:

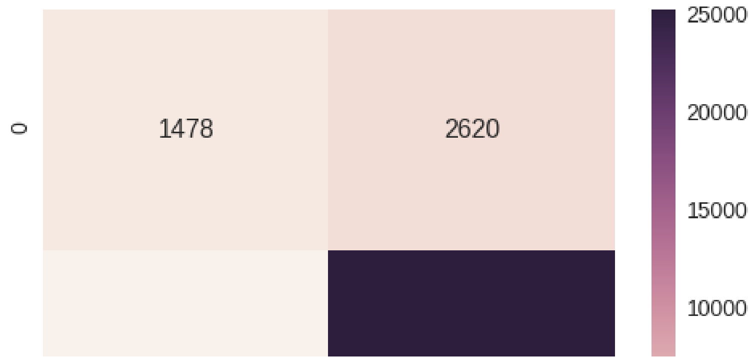
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f88bd747978>
```

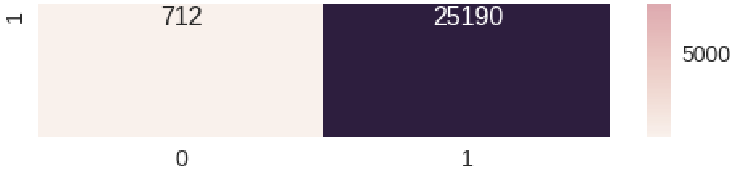


In [125]:

```
pred = (model.predict(X_test1))
print (classification_report(y_test,pred))
result = confusion_matrix(y_test,pred)
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
print(result)
sns.set(font_scale=1.4)#for label size
sns.heatmap(result, annot=True,annot_kws={"size": 16}, fmt='g')

print("f1 on test set: %0.3f%%"%(f1_score(y_test, pred,pos_label='positive')*100))
```

```
             precision    recall  f1-score   support

   negative       0.67      0.36      0.47      4098
   positive       0.91      0.97      0.94     25902

avg / total       0.87      0.89      0.87     30000

Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]

[[ 1478  2620]
 [  712 25190]]
f1 on test set: 93.797%
```





```
In [0]:
```

## LR with Grid search CV

- 1. Hyper parameter is 1
- 1. F1 score of Train data = 94.4%
- 1. F1 score of Test data
- 93.7%

## LR with Random search CV

Hyper parameter is 1

F1 score of Train data = 94.6%

F1 score of Test data

93.7%